A NOTE ON SPARSE COMPLETE SETS

by

Steven Fortune*

TR78-355

Department of Computer Science
Cornell University
Ithaca, New York 14853

*Research supported in part by the Office of Naval Research under Grant Number N00014-76-C-0018.

A NOTE ON SPARSE COMPLETE SETS

by

Steven Fortune *

Department of Computer Science
Cornell University
Ithaca, New York 14853

Abstract

Hartmanis and Berman have conjectured that all NP-complete sets are polynomial time isomorphic. A consequence of the conjecture is that there are no sparse NP-complete sets. We show that the existence of an NP-complete set whose complement is sparse implies P = NP. We also show that if there is a polynomial time reduction with sparse range to a PTAPE-complete set, then P=PTAPE.

Keywords: reduction, polynomial time, nondeterministic polynomial time, complete sets, sparsity

---

* Research supported in part by the Office of Naval Research under grant number N00014-76-C-0018.

1. Introduction

Hartmanis and L. Berman in [4] conjecture that all the NP-complete sets are isomorphic via polynomial time mappings. Of course, proving the conjecture would prove P≠NP and hence is likely to be hard to do. One consequence of the conjecture that they point out is that there could be no sparse NP-complete set, that is, there could be no NP-complete set having fewer than p(n) elements of length n, where p is a polynomial. A proof of this consequence could be viewed as evidence for the conjecture, but currently seems to be unobtainable, even under the assumption P≠NP.

In [2], P. Berman does obtain the following result. He shows that if there is a polynomial time reduction with sparse range mapping one NP-complete set to another, then P=NP. As a corollary, he shows that if there is an NP-complete set over 1^*, then P=NP. In this note we extend this result to show that if there is a sparse set complete for coNP, then P=NP. Thus, for example, if the set of tautologies can be reduced to a sparse set, then P=NP. We also show that if there is a polynomial time reduction with sparse range to a PTAPE-complete set then P=PTAPE.

The general idea of the proof is the following. We will give an algorithm to decide if a Boolean formula F written in conjunctive normal form is satisfiable. The running time will be polynomial under the assumption that there is a NP-complete set with sparse complement. The algorithm constructs a binary tree where the nodes are labeled with formulas obtained by assigning values to some of the variables in F. In general,

a node labeled by formula G will have two sons, one labeled with the formula obtained by setting one of the variables in G to 1, the other labeled with the formula obtained by setting the variable to 0. Of course, if such a tree were completely constructed, it would have exponential size. However, by using information gathered from a mapping to the cosparse complete set, the tree can be pruned to only a polynomial in size.

2. Sparsity of complete sets

In the proof of Theorem 1 we use the fact that SAT, the set of satisfiable formulas written in conjunctive normal form, is NP-complete. This was originally shown in [3]; the textbook [1] also contains a proof along with additional information on complete problems.

Theorem 1. Suppose there is an NP-complete set L which is cosparse, that is, there is at most a polynomial in n of words of length n not in L. Then P=NP.

Proof. Since L is NP-complete, there is a polynomial time computable function t such that F is in SAT if and only if t(F) is in L. The following algorithm will decide if a formula F is satisfiable.

```
Create the root node and label it with F.
WHILE the root is not marked "unsatisfiable" DO
   Pick the lowest node n in the tree not marked
   "unsatisfiable".
   Let the label of n be G.
   Choose a variable x appearing in G and create two sons
     of n.  Label one with the formula obtained by setting
     x=0 in G (and doing trivial simplifications:  y+0 = y,
     y+1 = 1, (y+z)·1  = y+z, (y+z)·0  = 0), label the
     other with the formula obtained by setting x=1.
   If there is a node corresponding to a satisfying assignm
     (i.e. a node labeled with the formula 1)
     THEN output ("satisfiable"); STOP.
   WHILE there is an unmarked node k with formula H satisfy
     either
     a)  both sons of k are marked "unsatisfiable"
     b)  H is trivially unsatisfiable (i.e. has a conjunct
         which is 0)
     c)  there is some node k' with formula H' marked "unsat-
         isfiable", and t(H) = t(H')
or   d)  some ancestor of k  is marked unsatisfiable
     DO  mark k "unsatisfiable" END
   END
output ("unsatisfiable")
```

The correctness of the algorithm follows from the assertion that a node is marked "unsatisfiable" only if in fact the formula of the node is unsatisfiable. This in turn follows by examining the four cases in which a node is marked "unsatisfiab

To see that the algorithm runs in polynomial time, first note that there are only polynomially many different values of t(H) not in L, as H varies over the formulas obtained by assign values to some of the variables in F. We will show that after

at most v iterations of the outer loop, where v is the number of variables in F, either a satisfying assignment is found or a new value of the range of t is discovered to be not in L. Hence the whole algorithm runs in polynomial time.

Consider a node n labeled with formula G chosen at the start of some iteration of the outer loop. Note that t(G) is not known not to be in L as n is unmarked. Suppose G has k variables. We will show by induction that after at most k iterations of the outer loop either a new value of the range of t is discovered to be not in L or a satisfying assignment is found. If k=1 then the two formulas assigned to the sons of n are variable free. Hence either at least one is the formula "1" and a satisfying assignment is found, or both are "0" and t(G) is discovered to be not in L. The inductive step, k>1, breaks into two cases. Either both formulas assigned to the sons of n are immediately marked "unsatisfiable" or at least one of them is not. In the former case node n will also be marked "unsatisfiable" and t(G) will be discovered to be not in L. In the latter case one of the unmarked sons will be chosen at the next iteration of the outer loop, as the son must be the lowest unmarked node in the tree. The induction hypothesis now applies since the formula of the chosen son has at most v-1 variables. Hence after at most another v-1 iterations either a new element of the range of t is discovered not to be in L, or a satisfying assignment is found. □

As another application of this technique, we have the following theorem. QBF here is the set of valid quantified Boolean formulas; it was shown to be PTAPE-complete in [5].

Theorem 2. Suppose there is a polynomial time computable function t and a set L such that

a) F is in QBF if and only if t(F) is in L

b) $|\{t(w): w \text{ is of length } n\}| < p(n)$ for some polynomial p.

Then P=PTAPE.

Corollary If there is a PTAPE-complete set over 1^*, then P=PTAPE.

Proof (of Theorem 2.) The following algorithm will decide whether a Boolean formula $F=\forall x_1 \exists x_2 \ldots Q x_v H(x_1,\ldots,x_v)$ is valid.

Create the root node and label it with F
WHILE the root is unmarked DO
  Pick the lowest unmarked node, n, in the tree.
  Let the formula labeling node n be G, and y the variable of the outermost quantifier.
  Create two sons of n. Label one with the formula obtained from G by setting y=0, the other with the formula obtained by setting y=1.
WHILE there is an unmarked node n with formula G satisfying one of
  a) The leading quantifier of G is $\exists$ and n has a son marked "valid"
  b) The leading quantifier of G is $\forall$ and n has a son marked "invalid"
  c) G is the formula 1
  d) G is the formula 0

```
    or  e) There is some marked node n' labeled with a formula
           G', and t(G)=t(G').
        DO
        In cases (a) or (c) mark n "valid".
        In cases (b) or (d) mark n "invalid".
        In case (e) mark n the same as n'.
        END
    END
Output the label of the root.
```

The proof of correctness is similar to that of Theorem 1. The running time analysis depend on the fact that the range of t is sparse and is otherwise analogous to that of Theorem 1. □

REFERENCES

[1] Aho, A., J. Hopcroft and J. Ullman. "The Design and Analysis of Computer Algorithms", Addison-Wesley Publishing Company, Reading, MA, 1974.

[2] P. Berman. "Relationships Between Density and Deterministic Complexity of NP-complete Languages", Fifth International Colloquium on Automata, Languages and Programming. (1978) pp. 63-71.

[3] Cook, S. "The Complexity of Theorem-proving Procedures", Proceedings of the Third Annual ACM Symposium on Theory of Computation (1971), ppg. 151-158.

[4] J. Hartmanis and L. Berman. "On Isomorphisms and Density of NP and Other Complete Sets", Proceedings of the Eighth Annual ACM Symposium on Theory of Computing (1976) pp. 30-40, also in SIAM J. Computing Volume 6, No. 2, June 1977, pp. 305-322.

[5] Meyer, A.R. and L.J. Stockmeyer. "Word Problems Requiring Exponential Time", Proceedings of the Fifth Annual ACM Symposium on Theory of Computing (1973), pp. 1-9.